Temporal IRAD Preliminary Design and Findings

Part One: Design Choices and Rationales

4 May 1995

Intergraph Corporation

For questions and comments, please contact:

Gail Langran Kucera
604-360-2655
kucera@islandnet.com

This report describes the work to date on Intergraph's Internal R&D (IRAD) project on spatiotemporal modeling for geographic decision support, commonly referred to as the "Temporal IRAD." The work is described in two parts, released as separate reports. Part One (this report) documents the reasoning behind the many choices that have led to a provisional design. Part Two (to be issued shortly) describes the provisional design itself and the prototyping plans. A final report will be released upon completion of the prototype.

Gail Kucera, the IRAD's principle investigator, authored this report. However, the thoughts of many are represented here through literature review and lengthy discussions internal to Intergraph. In particular, Harold McDaniel, Janet Conklin, and Sam Bacharach were generous with their expertise and advice, which was gratefully received and incorporated within.

Temporal IRAD Preliminary Design and Findings

Part One: Design choices and rationales

Raster vs. vector
Temporal types and operators
Versioning method
Management of persistent feature identities
Describing and maintaining temporal data quality
Management of uncertainty

This report describes some of the issues that were considered in the design of the Temporal IRAD prototype. A second report follows that describes the design of the prototype system itself.

The issues addressed in the sections below are: whether to prototype a raster or a vector temporal system; how to manage dates, times, and their manipulation; at what level of detail vs. redundancy to maintain feature histories; how to describe changes to features that are so extensive they can be considered a new feature; description of temporal data quality; and management of uncertainty over time.

1. Raster vs. vector

We have chosen to prototype temporal vector rather than temporal raster data structures after a complete review of the literature of spatiotemporal analysis. One factor was of overriding importance in this decision: raster data are not naturally feature-based, and feature analysis is among the strongest requirements of the most sophisticated geographic applications, including TFG.

To attain feature-based raster analysis, we would need to design a feature look-up table that would index features within the raster grid. The look-up table would manage uncertainty, versioning, and identity persistence. To ensure good spatial referencing, the look-up table could end up assuming many of the characteristics of vector data structures. The natural simplicity and efficiency of raster data--its best trait--would be negated by such overhead.

Other factors that played a role in our decision were as follows.

- Intergraph's vector data structures are extremely flexible and extensible, particularly within the object-oriented environment available to us.

- Most existing work in temporality has focused on temporal vectors, which means that this project can benefit from the thoughts of others.

- Intergraph has implemented spatiotemporal vector systems for niche applications, which permits this project to draw on that experience.

- Although raster data structures can be extended via fat pixels, pointers-to-buckets, temporal quadtrees, compressed snapshots, or other methods, these techniques are completely untried in the temporal domain, would require a considerable amount of programming, and may entail more risk than the better-proven vector extensions.

- Within TFG, the vector database is functionally the better place to maintain temporal data. This is likely to be true of many dual raster / vector systems.

2. Temporal types and operators

A fundamental requirement for our temporal system will be a means of expressing timestamps, and a means of performing operations using timestamps. The long-range goal is to accommodate the date / time types and the temporal operators being defined as part of the SQL / MM (SQL Multimedia) definition, which is part of ISO's SQL 3 initiative. SQL / MM presently has a draft statement of temporal operators that would be recognized as standards within a database system.

The temporal operators of SQL / MM were extended by Kucera[1] from Allen's[2] work that identified all possible relationships between two intervals located on a timeline. Allen's "interval-based" temporal logic (Figure 1a) is the goal for any temporal system, since all feature occurrences occupy an interval on the timeline at some resolution. Because this IRAD project is brief, however, we will implement a "point-based" temporal logic only (Figure 1b).

Implementing point-based logic rather than both logics reduces the number of temporal operators considerably and permits us to focus our efforts on research issues rather than programming of known requirements. In addition, the point-based operators are building-blocks for the interval-based operators. Note that all existing logics assume that timestamps are crisp, not fuzzy. Permitting fuzziness at either or both ends of a temporal interval increases the number of operators substantially.

[1]Kucera, Henry A. "Extending SQL for Geographic Information: SQL / MM." Unpublished MSc Thesis, Department of Geography, University of Victoria, Victoria, BC.
[2]Allen, J. F. (1983). "Maintaining Knowledge about Temporal Intervals." *Communications of the ACM* 26, 11, 832-843.

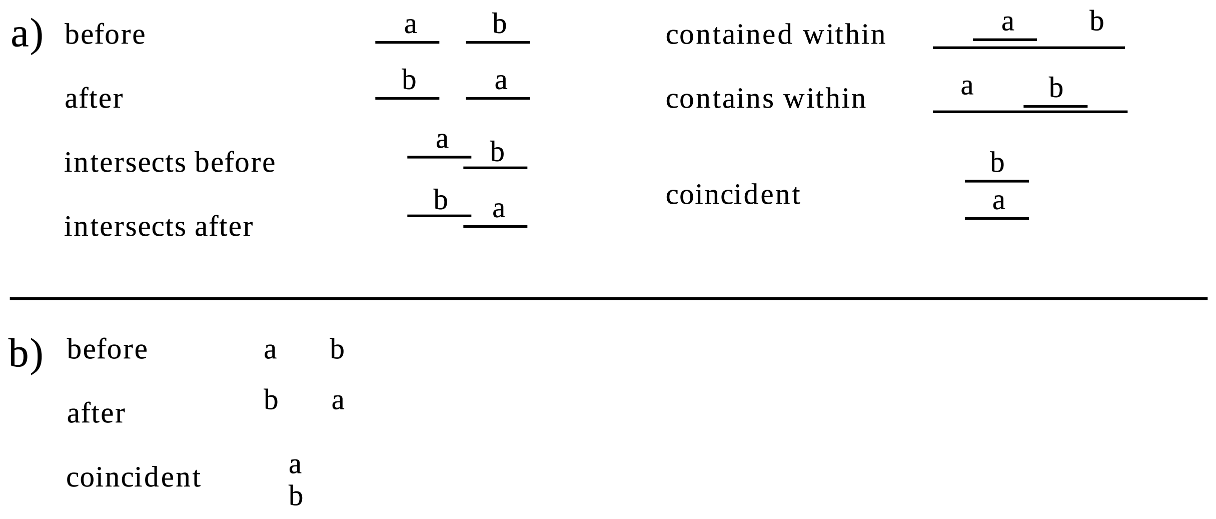

Figure 1. Examples of interval-based vs. point-based temporal relationships. a) Interval-based temporal logic describes crisp relationships between intervals on the timeline. b) Point-based logic describes crisp relationships between points on the timeline.

An alternative to implementing a date type does exist. Instead, we could create a "timestamp catalog" to save some compute time and storage space. Each new timestamp required within the system would be logged in the timestamp catalog, receiving a sequential timestamp number. The timestamp number would be stored in feature records, rather than the timestamp itself. The benefit is that the timestamp number would require fewer bytes to describe than would a timestamp. If timestamps are stored at the attribute level (as in the prototype), the benefit multiplies. The benefit fades if temporal resolution is high or the timespan handled by the system is long, since then catalog numbers would require more bytes. We are shelving the idea of a timestamp catalog for now, but retaining it as a possible measure to be re-examined as we progress.

3. Versioning method

A temporal system works by creating new versions as change occurs. One must choose what entities are "versioned" (i.e., have new versions created) and under what conditions. Versioning can be performed at several levels, depending on the requirements. A new version can be created each time any aspect of a feature changes (Figure 2a). Or, versions can be created independently for geometric changes or changes to the attribute set (Figure 2b). Finally, versions can be created independently for each attribute in the attribute set (Figure 2c).

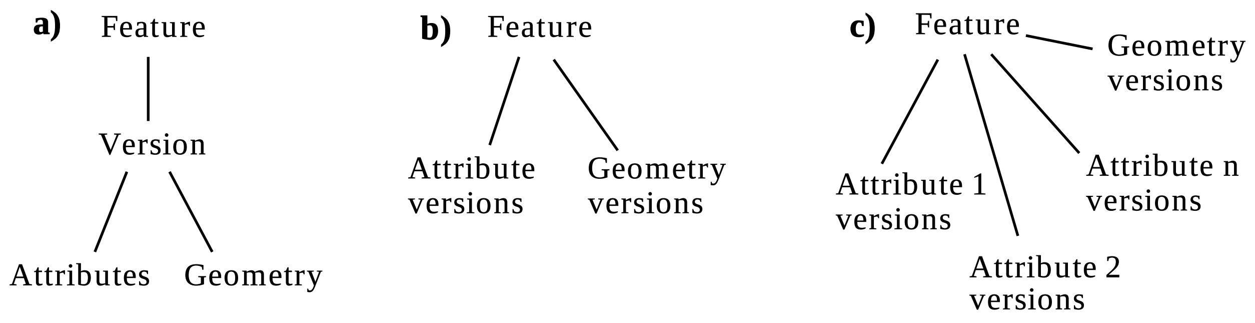

Figure 2. Versioning at different levels. a) A new version is created each time any aspect of the feature changes. b) Versions are created for geometric changes, and independently for the attribute set. c) Versions are created independently for each individual attribute.

How versioning is performed has a major impact on the efficiency and speed of the temporal system. Table 1 summarizes the pros and cons for each type of versioning. A university prototype sponsored by Intergraph[3] showed Option A (Figure 2a) to be by far the fastest in response time. For that reason, we have chosen it as the model for our temporal prototype.

Table 1. Pros and cons of versioning at different levels.

	Pro	Con
a) **Feature-level versioning**		
Simplest to implement	❖	-
Fastest retrieval	❖	-
Most wasteful of storage	-	❖
b) **Independent attribute/geometry versions**		
Good if geometry changes are independent (not an issue with Intergraph configuration)	-	-
c) **Attribute-level versioning**		
Good if attribute changes are independent	-	-
Very slow retrieval times	-	❖
Most thrifty of storage	❖	-
Most difficult to implement	-	❖

Our choice of versioning at the feature level creates a new version every time an attribute changes. Thus, it uses redundancy (the unchanged attributes appear in each feature version along with the changed ones) to enhance system performance. Figure 3 is a conceptual view of how feature-level versioning works. In the example, Feature A is observed at four different moments in time; the final observation records a change. The database describes the history of Feature A with one version that is effective for T1 until T4, and a second version that becomes effective at T4 until a new change is observed. Of Feature A's three attributes, only one ("composition") changes. The others are stored redundantly in both versions. This seemingly inefficient storage strategy

[3]Bedard, Yvan (1993). "Theoretical Aspects to Deal with Temporal Data within MGE-Dynamo." Unpublished research report, Centre de recherche en géomatique, Université Laval, Ste-Foy, Quebec.

permits an efficient retrieval strategy: the system only needs to find the effective feature version to find all effective attribute versions, rather than seeking them independently.

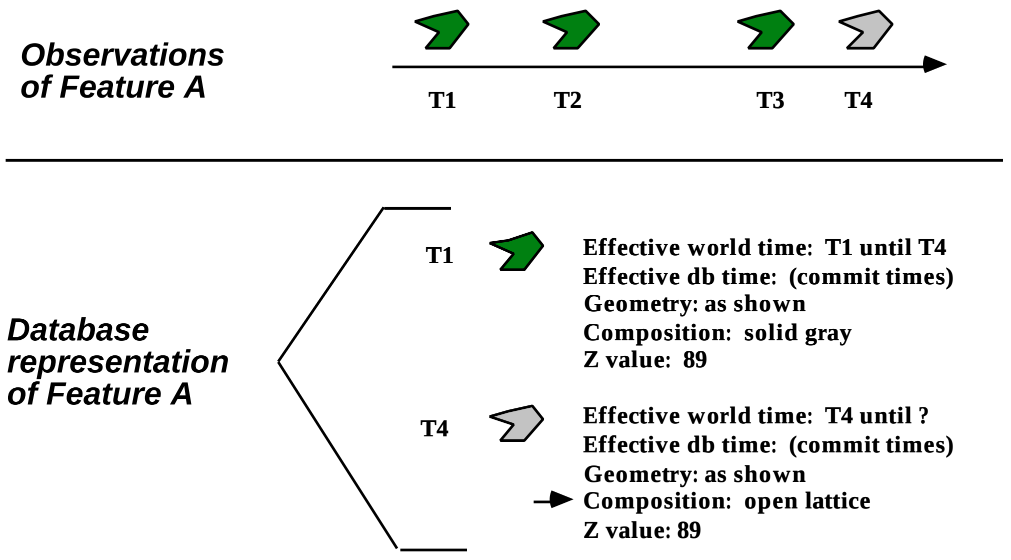
Observations of Feature A
T1
T2
T3
T4
Database representation of Feature A
T1
Effective world time: T1 until T4
Effective db time: (commit times)
Geometry: as shown
Composition: solid gray
Z value: 89
T4
Effective world time: T4 until ?
Effective db time: (commit times)
Geometry: as shown
Composition: open lattice
Z value: 89

Figure 3. Conceptual view of feature-level versioning. In the example, a new database copy of Feature A is created and stored when the color fill attribute changes. This means that Geometry and Z-value are stored redundantly but ensures the best possible retrieval times.

We have read with interest the useful report provided by PSR (number 2549, on "Message Reporting on Changes to Terrain and Features"). We had arrived at some of the same conclusions independently and are pleased with the convergence of views. One exception is the method of describing temporary or overdue change data. Because our temporal prototype will keep a version for each change, including temporary changes (discussed on pages 7 and 8 of the PSR report), we can trigger the generation of a new version after the correct duration of the old one has elapsed. Thus, we will store a temporal decay value as a timestamp that the system monitors for expiration, not as a class value (short, intermediate, and long) as suggested by PSR. In addition, we plan to maintain attributes that describe the changes contained in versions of features, thereby avoiding the use of annotation fields that cannot be analyzed automatically.

4. Management of persistent feature identities

A temporal system is concerned with more than how current are its data; it also attempts to maintain a history of the information as it changed over time. A feature-based geographic system is concerned with feature histories. Thus, the goal is to describe each feature's birth, life, and death, and its geometric, topological, and thematic characteristics during that timespan. Features are born and die when they appear and disappear. Features also can die and be reborn as new features when they change in certain ways. A feature that changes but remains the same feature has a *persistent* *identity*. A feature that changes into another feature through change to its characteristics undergoes an *identity change*.

It is critical to determine for each feature how it can change and still be considered the same feature. For example, a lighthouse can be painted a new color and still be considered the same lighthouse. If its light is changed, the lighthouse itself retains its essential identity. More than likely, if the structure is torn down and rebuilt in the same location, it still would be considered the same lighthouse by most applications. But what if it were moved 50 meters? Or 50 kilometers? Evidently, a change in location by more than a certain amount causes an identity change to a lighthouse, since most applications would consider a 50 km move to have killed the original lighthouse and created a new one.

A temporal system that automates updates (as opposed to one where an analyst oversees versioning) must pre-establish what comprises persistent identity vs. identity change for each type of update to a feature. Table 2 summarizes the elements of feature birth and death.

Table 2. Ways that features are born and die.

Change	**death**	**birth**	
Appearance		❖	A new feature appears.
Disappearance	❖		An existing feature vanishes.
Identity attribute	❖	❖	Some attributes govern identity. Any change whatsoever to an ***Identity Attribute*** means the existing feature has died and a new one is born. The "feature type" attribute (in TFG, the FACC code) is an Identity Attribute.
Domain attribute	❖	❖	Some attributes can change in certain ways without interrupting identity persistence, while other changes cause identity change. Movement by more than a given distance is one case where domain change causes death of one feature and birth of a new one.
Spatial context	❖	❖	Topological changes cause a feature to no longer serve its original purpose, making it a new feature rather than a new version of the old one. A bridge spanning a different river or connecting different road segments is a n example of spatial context causing death and birth.

At minimum, a temporal system must implement identity management by establishing identity attributes, domain attributes, and spatial context indicators for all its features. The establishment of identity attributes and other terminators of feature persistence is application-dependent. For the temporal prototype, the identity attribute for all features is the FACC code: if the FACC code changes, the old feature has died and a new one is born. We will also establish a limited number of domain attributes and spatial context indicators for demonstration purposes. For example, we might choose to consider change in wetness to a swamp to be cause to consider the original swamp to have died and a new vegetation area to have been born.

We will implement identity management through the use of triggers. Triggers use "listening objects," which await a state change to a specified data value and run a method when the awaited state change occurs. A simple feature identity trigger would operate as follows.

State change	**Method triggered**
FACC changes	Add death timestamp to existing feature Create new feature, add birth timestamp, create first version

A more sophisticated trigger also could check whether the FACC change is a legal one (i.e., one whether the data dictionary describes that change as possible, or as a probable error). If a probable error exists, the system warns the interactive user or queue the problem update for interactive resolution.

A second type of trigger will be needed to govern the creation of new versions for existing features. Table 3 enumerates how new versions of existing features are created. In most cases, any change at all to non-identity attributes is cause for creating a new version. For prototype purposes, we will consider addition of an attribute to an empty attribute field not to be cause for creating a new version, although this assumption might be changed for specific applications. We will not prototype the case where small changes to attributes that are below given tolerances are not considered to be cause for creating a new version.

Table 3. Ways that new versions are created.

	New version?	
Change to "versioning attribute"	***Y***	Any change whatsoever to this attribute causes a new version to be created.
Population of empty attribute	***N***	An unpopulated attribute becoming populated does not cause a new version to be created.
Tolerance change	***N***	A change within a given tolerance may not cause a new version to be created. The prototype IRAD system will not implement this type of versioning.

Identity attributes are owned by a feature, as are atemporal attributes that are assumed never to change or whose changes are not tracked temporally by the system. Conversely, domain attributes and all versioning attributes are owned by feature versions (Figure 4).

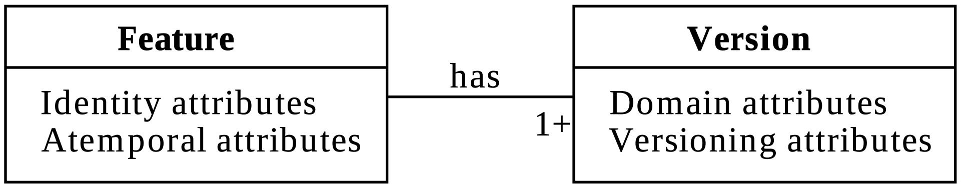

Figure 4. Ownership of attributes by a feature and its versions. Each feature has at least one version (the one it was born as). Identity attributes whose change cause the feature to die are owned by the feature, as are atemporal attributes that are assumed never to change or whose changes are not tracked temporally by the system. Each version has domain attributes, where certain changes trigger feature death; and versioning attributes as described in Table 3.

5. Describing and maintaining temporal data quality

Data quality is among the most important concerns of civilian spatial systems. Civilian users are concerned with:

- accountability of decision-makers, which may involve legal liability and court challenge;
- delaying decisions until the data quality can support good ones;
- "truth in advertising," where risk factors in poor data are known.

In a military system, the first two concerns are lesser, and the last (knowledge of risk factors) is greater. Because military decision-makers must know how steady is the foundation upon which their decisions are based, we will describe temporal data quality in the prototype with the same care as it would be described in a civilian system. It is easy to jettison unnecessary indicators later if they are wasteful or burdensome.

Components of data quality description include lineage, completeness, logical consistency, resolution, and accuracy. The discussion that follows describes how each of these indicators applies to the temporal domain and how it will be implemented in the prototype. A later section discusses how certainty, which also pertains to data quality, will be addressed.

5.1 Lineage

Lineage is provided by an ID code that references a catalog entry that describes the source document and the data conversion process at a level of detail required by the application. We can provide the best lineage by assigning this value to *each attribute* of a feature, since we assume that different attributes will be collected from different source documents at different times. The IRAD prototype will follow this strategy, while retaining the option of having only one lineage value for one feature in later prototypes if this level of lineage is deemed excessive.

5.2 Completeness

Temporal completeness ensures that no gaps in the temporal data exist, providing a description of each feature at all moments in time (even if the description is merely "inactive" or "unknown"). Completeness in spatial data is provided by explicit topological description that ensures that all space is accounted for. A similar technique can be used to describe the temporal topology of each feature. In the case of the prototype, each feature's temporal topology is a line with nodes at moments of change and arcs at intervals of stability (Figure 5). Thus, explicit temporal topological description seems excessive. Section 6 of this discussion paper (below) illustrates a case, not applicable here, where explicit temporal topological description could be warranted.

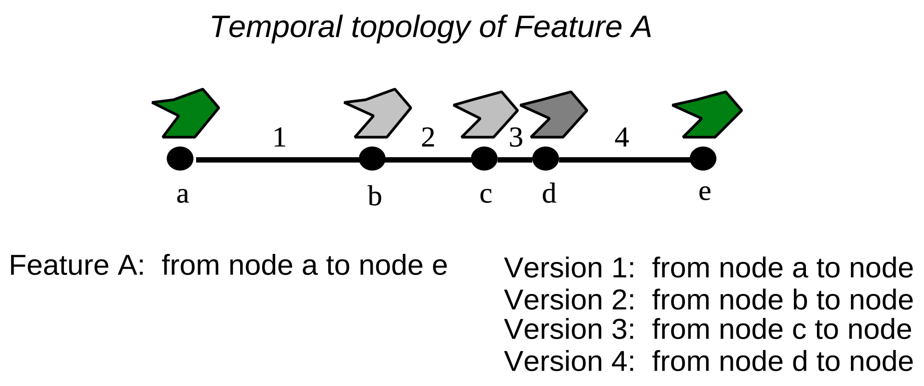
Temporal topology of Feature A
1
2
3
4
a
b
c
d
e
Feature A: from node a to node e
Version 1: from node a to node
Version 2: from node b to node
Version 3: from node c to node
Version 4: from node d to node

Figure 5. Temporal topology of a feature. Topological description can be explicit or implicit. Testing for topological completeness can be done at time of versioning, birth, and death, or can be external subroutines to be used as needed.

The prototype system will check for topological completeness at the time a new feature is born, an old one dies, or a new version of an existing feature is created. The tests will ensure that all moments are filled, and that no two versions occupy the same moment.

5.3 Logical consistency

If no two versions occupy the same moment, a feature's temporal data are logically consistent. Temporal inconsistencies also can cause problems within one level of resolution and between levels of resolution. Within one level of resolution, some data will be more current than others. Ideally, analytical procedures will be weighted to take advantage of certainty and currency values stored in the metadata. Between levels of resolution, a feature may be updated at one level but not another. TFG's requirements include not "leaking" information from one level to the next. Thus, linkages and metadata will be critical to describe temporal inconsistencies.

5.4 Resolution

Temporal resolution is a function of sampling interval. For TFG and most other spatial systems, sampling intervals often will be erratic. Thus, it is critical to store information that describes observations when features did not change, as well as observations when features did change, as a way of narrowing the period between samples when the feature change may have actually occurred (Figure 6).

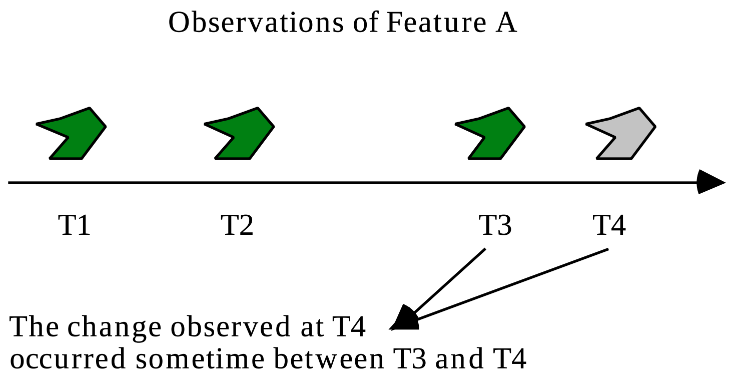
Observations of Feature A
T1
T2
T3
T4
The change observed at T4
occurred sometime between T3 and T4

Figure 6. The system must help to narrow the time interval when an observed change may have occurred. Most temporal systems record only moments of change. In such systems, the change observed at T4 could be suspected of occurring anywhere between T1 and T4.

In the prototype temporal system, we will store moments of observed change. We also will store an "until date" timestamp for each version, which is the time of the latest forward observation of that version before it changed to the next (Table 4). Note that the "until date" differs from the "end date" used by many temporal systems. An "end date" of one version equals the "start date" of the next. In contrast, the "until date" of one version can be earlier than the "start date" of the next version, since we assume that our observations do not always coincide with precise moments of change. In cases where we can assume our observations coincide with moments of change, the "until date" is adjusted accordingly.

Table 4. Describing the latest forward observation to narrow the period when a change may have occurred. Using the example of Figure 3, the prototype's "until date" would work as shown here.

Versions of Feature A

Time	Version	Start date	Until date
T1	Version 1	T1	T1
T2	Version 1	T1	T2
T3	Version 1	T1	T3
T4	Version 1	T1	T3
	Version 2	T4	T4

5.5 Accuracy

The temporal prototype will not embed any quality information on temporal accuracy. We assume that different source data will arrive with different calibers of timestamps based on how they were collected. This is easily handled via a source catalog, but is excluded from the prototype so we can focus on more challenging issues of temporality.

6. Management of uncertainty

Two types of uncertainty concern us: uncertainty in feature identification, and uncertainty in identifying a feature as a changed version of a previously existing feature.

Uncertainty in feature identification is resolved through improving feature extraction algorithms. The TFG system will assign one probability value for each feature to describe the certainty that a feature is correctly identified. The prototype temporal system will provide a probability value for each attribute of each feature to describe the certainty that the attribute is correct.

The second type of uncertainty, which involves matching a changed feature to its previous version, requires algorithms to determine how probable the change is in the timespan allotted. For static features, the process should be relatively straightforward. Because all TFG features are relatively static, the Phase One prototype assumes that uncertainty in matches between versions of features can be resolved before new data are committed.

If moving features are added to the system, the version-matching process becomes far more uncertain. Imagine two identical features A and B, and a new source document that shows a third identical feature that is within the traveling range of both the original features (neither of which are in evidence) (Figure 7). The newly identified feature could be a new version of A, or B, or a new feature (C) entirely. Thus, the linkages between versions would need to be assigned probability values.

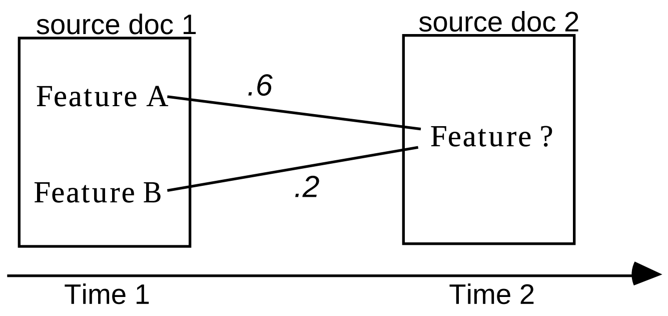

Figure 7. Uncertainty in identifying new versions of existing features. Feature A at Time 2 has a .6 likelihood that it has the attributes of the new feature in source document 2, and (by extension) a .4 likelihood that it has the attributes of source document 1.

The temporal prototype will assume that no uncertainty exists in identifying new versions of existing features, i.e., a temporal topology where each moment holds one definitive state (Figure 8a). The alternative is to permit features to be identified and linked with uncertainty, exhibiting a branching temporal topology (Figure 8b), which is a research project in and of itself. To support branching temporal topology would require more complex temporal data structures and the design of integrity rules to govern such questions as:

- Must the certainty values for a new source feature's identity add up to 1?
- Must the certainty values at each temporal node add up to 1?
- Must the certainty values in ascending temporal nodes descend?

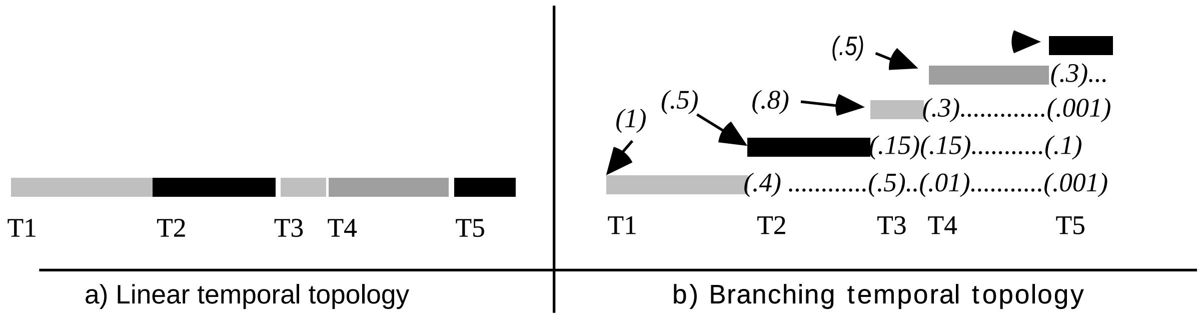

Figure 8. Linear and branching temporal topologies. a) Linear temporal topology assumes that no uncertainty exists in linking versions of features. b) Branching temporal topology permits uncertain identification of versions and assigns a probability that the linkage is correct. In the example, the feature has a 0.5 probability of having undergone a state change at T2, a 0.4 probability of having remained the same, and a 0.1 probability of having neither of these cases be true.

7. Summary

This report documents the fundamental choices that underlie the design of the temporal prototype. A second report will be issued within the month to document the initial design of the temporal prototype. Since the prototype represents research, not development, we expect the design to change in the course of the prototyping. We will document the changes and the final design in a final report to be delivered with the prototype.